Forecasting NBA Regular-Season Outcomes: Spread, Total, and Offensive Rebounds

A Reproducible Master Dataset and MAE-Based Evaluation using Pre-Game Features

Group 2
Jason Lee
Ethan Yountz
Rujula Yete
Olivia Liu
Seth Hall

STOR 538: Sports Analytics
Professor Mario Giacomazzo

Data Information

Data Source and Sample Selection

This project evaluates predictive models for three numeric outcomes of NBA games—Spread, Total, and offensive rebounds (OREB)—with predictive accuracy assessed using mean absolute error. To ensure that all members of the group could work from a shared data foundation regardless of their chosen target variable, we constructed a single master dataset with one row per game. We aligned home-team and away-team statistics in the same observation. This structure is particularly appropriate for modeling Spread, since Spread is inherently a home-versus-away comparison, and it is equally useful for Total and OREB because it stores both teams' contributions needed to compute combined outcomes.

The primary source for the master dataset is the TeamStatistics box-score database, which provides team-level game statistics for the NBA from the 1947 season through the present day. Although the raw source is historical in scope, the objective of this project is not long-run description but forecasting regular-season games in March–April 2026. For that reason, we restricted the modeling sample to the modern post-COVID era beginning with the 2021–2022 regular season (October 19, 2021) and excluded the COVID/bubble season and all earlier seasons. This restriction is motivated by the fact that the NBA's statistical environment is not constant over long periods: the league's shot profile, three-point volume, spacing, rotation patterns, and strategic norms have evolved substantially. Using a more recent and stylistically consistent sample is therefore more likely to produce relationships that generalize to the 2026 games we are required to predict.

Elo Ratings as a Measure of Team Strength

In addition to box-score data, we merged an NBA Elo dataset (nba_elo.csv) as a compact, pregame measure of team strength. The Elo rating system used in this dataset was developed by Neil Paine, a sports analyst formerly associated with FiveThirtyEight (538), and the dataset was obtained from his publicly available GitHub repository. Elo rating systems are widely used in sports analytics because they provide a principled way to summarize team quality using only prior outcomes while accounting for opponent strength: in an Elo framework, a team's rating increases when it wins—especially against strong opponents—and decreases when it loses—especially against weaker opponents. This produces a continuously updating estimate of relative team ability that is interpretable, comparable across teams, and empirically effective for forecasting win probabilities and score differentials.

In our dataset, Elo enters as HomeElo_pre and AwayElo_pre, which represent each team's rating prior to the game, and as EloDiff = HomeElo_pre − AwayElo_pre, which directly captures a pregame strength gap that is conceptually aligned with Spread prediction. Elo is also respected in applied work because it often performs competitively as a baseline model, can be combined with richer box-score features without overfitting, and supports transparent interpretation (e.g., larger Elo differences correspond to stronger expected performance). When Elo was not available for a given game, we retained the observation and left Elo fields missing rather than deleting the game, ensuring that Elo coverage did not determine which games entered the modeling sample.

Construction of the Game-Level Dataset

The TeamStatistics source is recorded at the team-game level, so each contest appears twice—once for each team. To create a modeling dataset with a single observation per game, we reshaped the data into a home–away paired format. Concretely, we separated home-team rows from away-team rows using the home indicator, renamed all box-score columns with Home_ and Away_ prefixes, and then merged the two records for each game using the game identifier. This produced a game-level table in which both teams' statistics are aligned within the same row, enabling direct construction of matchup predictors such as home–away differences.

Definition of Outcome Variables

From the aligned box-score totals, we computed the three outcomes required for the project. Spread captures the score differential between teams, Total represents combined points scored, and TotalOREB represents combined offensive rebounds:

$$Spread = HomePTS - AwayPTS$$
$$Total = HomePTS + AwayPTS$$
$$TotalOREB = HomeOREB + AwayOREB$$

These variables are stored in MasterDataset.csv as Spread, Total, and TotalOREB.

Feature Engineering

To support prediction of all three outcomes, we augmented the raw home and away box-score fields with a set of engineered predictors that summarize efficiency, tempo, scheduling, and recent form. First, for many paired statistics we constructed difference features of the form diff_X = Home_X − Away_X, which are especially relevant for Spread because they quantify relative advantages (e.g., turnover differential, rebound differential, shooting differential). Secondly, we derived standard shooting efficiency measures from box-score components. Effective field-goal percentage and true shooting percentage were computed as:

$$eFG = (FGM + 0.5 \cdot 3PM)/FGA$$
$$TS = PTS/\ (2(FGA \cdot 0.44 \cdot FTA)$$

We also included rate-style features that normalize performance across games (e.g., three-point attempt rate $3PA/FGA$, free-throw rate FTA/FGA, and turnover rate $TOV/\ (FGA + 0.44 \cdot FTA + TOV)$. Because the dataset includes both offensive and defensive rebounds for each team, we additionally computed an offensive rebound rate proxy using opponent defensive rebounds: $OREB\_rate = OREB/\ (OREB + OppDREB)$. Third, because Total scoring is strongly influenced by tempo, we estimated possessions as a pace proxy: $Poss \approx FGA - OREB + TOV + 0.44 \cdot$ FTA, and retained home, away, and game-level summaries (Home_poss, Away_poss, poss_avg, poss_diff). Additionally, we added scheduling context through rest and back-to-back indicators computed from each team's prior game date (Home_rest_days, Away_rest_days, and back-to-back flags), along with a rest differential. Finally, to capture "form" while avoiding look-ahead bias, we created lagged rolling averages over recent games (5- and 10-game windows) for points scored, points allowed, and margin, and included the corresponding home–away differences.

Lastly, we included exponentially weighted moving average (EWM) variables to capture recent scoring trends while placing greater weight on more recent games. Variables such as comb_ewm20_total and comb_ewm10_total represent exponentially weighted averages of game totals over the previous 20 or 10 games for each team. Compared with simple rolling averages, EWM statistics respond more quickly to

changes in team style or pace because older observations receive less weight. These values were computed from each team's chronological game history (combining home and away appearances) and shifted by one game so that only pregame information was used. The home and away values were then summed to create matchup-level predictors such as comb_ewm20_total, which summarize the recent scoring tendencies of both teams entering the game.

Spread Methodology

Overview & Approach

For all spread analysis, we chose to start our training and testing periods at the beginning of the master dataset (10/19/2021). We knew that we wanted to remove all observations before the bubble because we identified that the fast-paced evolution of the NBA would cause prior data to not be as accurate for predicting the style and officiating of today's game. Stats from the dead ball era, for example, could cause systematic low spread values because of the slower pace of play and the totals from that time. Models were tested on a 5-fold CV to tune parameters and hyperparameters without lookahead bias and a held-out test set of January 14th - February 19th, with features frozen to January 14th values to mimic the final prediction requirements.

Baseline Model

The naive baseline we used was the average margin for the home team minus the average margin for the away team, divided by 2. We chose this method to simply estimate team strengths and to have an error value to improve upon. Analysis of this naive approach over the selected period showed a CV MAE of 11.8681 and a test MAE of 12.436.

The base features that we ultimately selected were elo_diff, netrating_diff, offRating_diff, defRating_diff, pace_diff, diff_last5_margin, rest_diff, and home indicator. We felt that this would present us with a good baseline of relative team strength, recent form, and scheduling factors like location and fatigue from little rest. Additionally, pace was included, as teams that play faster will exaggerate their advantage or disadvantage in the final spread through more possessions. We ran a linear regression with those eight base features with the same split and got a CV MAE of 11.2246 and a test MAE of 14.3503 (features became stale very quickly).

Feature Selection

Since we were predicting future games, all features chosen for modeling would have to be available prior to tip-off. Features were selected to consider a team's form, scheduling factors, and historical performance to generate a final spread value.

For feature selection, we decided to use lasso-based regularization and to run an alpha sweep to help us balance the relationship between test MAE and the number of kept variables.

- **Core strength features:** elo_diff, netrating_diff, offRating_diff, defRating_diff, pace_diff, home_indicator
- **Recent form features:** diff_last5_margin, diff_last10_margin, diff_last5_pts_for, diff_last5_pts_against, diff_last10_pts_for, diff_last10_pts_against
- **Schedule/context features:** rest_diff, Home_b2b, Away_b2b, home_rest_days_capped, away_rest_days_capped
- **Season and matchup features:** winpct_diff_calc, season_margin_diff_calc, split_margin_diff_calc, h2h_margin_for_home

- **Strength-based Interaction terms:** home_indicator * elo_diff, home_indicator * netrating_diff, diff_last10_margin * elo_diff, diff_last10_margin * netrating_diff
- **Matchup Type Interaction terms:** diff_roll20_3pa_rate * diff_roll20_ts, netrating_diff * split_margin_diff_calc, diff_roll20_tov_rate * defRating_diff, diff_roll20_ts * split_margin_diff_calc, home_indicator * away_rest_days_capped
- **Lagged Efficiency Features:** diff_eFG, diff_ts, diff_tov, diff_oreb, diff_3pa_rate, diff_ftr

For the lagged features, we ran a lasso regression with a window of 5, 10, 20, and 30, and got back the same efficiency features (3pa rate and ts). When fitting the linear model, we tried the same rolling windows with the selected efficiency features and found a lagged value of 20 was the best for MAE. We also tried recency weighting for the lagged features, but it added little value in comparison to an even weight for the last 20 games.

Selected Features

After evaluating the relationship between kept variables and held-out MAE, we ultimately decided on using a lasso regression with an alpha of 0.15 to select the most relevant features for use in future models.

- elo_diff
- Home_indicator
- Diff_last10_margin
- Netrating_diff
- Split_margin_diff_calc (Home vs Away splits for both teams)
- Away_rest_days_capped
- Home_rest_days_capped
- Diff_roll20_tov_rate
- Home_b2b
- defRating_diff
- Winpct_diff_calc (Current season winning percentage)
- Diff_roll20_ts
- Diff_roll20_3pa_rate
- H2h_margin_for_home (past matchups between the two teams)
- Away_b2b

Further Modeling

Since there were a few collinear features, we decided to plug the selected features into a ridge regression to see if more signal could be derived from the correlated net / elo / rating variables.

Ridge Regression: Test MAE = 11.5082 with alpha = 1. We ran a sweep of 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100 - no substantial MAE improvement was found, so we kept an alpha of 1.

We then began using gradient boosting and XGBoost to test whether nonlinear relationships and higher-order interactions in the feature set could improve prediction beyond a linear baseline like ridge regression.

After running a hyperparameter sweep, the result was an XGBoost model with a test MAE of 12.0250, which will be discussed in the best model section.

Other attempts that failed to reach that error value were a ridge regression baseline with XGBoost-based residual correction, elastic net, and MLP.

Model	CV MAE	Test MAE
XGBoost	10.989	12.0250
MLP 64 -> 32	11.0920	12.0438
Ridge alpha=1	11.0053	12.0629
GradientBoost	11.0266	12.067
ElasticNet alpha=0.3	11.0187	12.0744
Ridge + XGBoost residual	10.9925	12.0992
Base linear model	11.2246	14.3503
Naive	11.8681	12.4360

Best Model

XGBoost produced the best performance of the models tested, in both CV MAE and Test MAE - showing it was good across folds and for our use case. We felt that the greater complexity in explaining the model was worth it for the increased performance.

To find a good configuration, a hyperparameter sweep was performed to balance model flexibility with strong regularization. Because the dataset of historical games that we thought were relevant is not extremely large, the goal was to avoid overly complex trees that could overfit the training data.

Full Sweep:

- n_estimators: 120, 160, 220, 300
- learning_rate: 0.02, 0.03, 0.05
- max_depth: 2, 3
- min_child_weight: 15, 25, 40
- subsample: 0.55, 0.65, 0.80
- colsample_bytree: 0.55, 0.70, 0.85

The final model used the following parameters/hyperparameters:

- n_estimators = 160 - sets the number of trees the model builds
- learning_rate = 0.03 - low learning rate trees do not quickly shift
- max_depth = 2 - each tree can only make two splits (reduce overfit)
- min_child_weight = 25 - min number of observations before a split
- subsample = 0.55 - each tree trains on 55% of the training data
- colsample_bytree = 0.55 - each tree only considers 55% of the features
- reg_lambda = 10 - L2 regularization value (strong penalty - shrinks values)
- reg_alpha = 2 - L1 regularization (sets values to 0)
- gamma = 1 - how much improvement is required for a split

With these features and importance values (portion of the total-split improvement from each feature)

Feature	**Importance**	**Description**
netrating_diff	0.3061	Difference in net rating before the game
winpct_diff_calc	0.1626	Difference in current season win percentage
split_margin_diff_calc	0.1500	Uses home and away splits to capture team habits
diff_last10_margin	0.0932	Last 10 margins to capture team form
defRating_diff	0.0560	Difference in the def rating before the game
diff_roll20_tov_rate	0.0363	Difference in turnover percentage over the last 20 games
home_rest_days_capped	0.0336	Number of rest days (0 - 4)
h2h_margin_for_home	0.0335	The previous 5 margins for the same two teams
diff_roll20_ts	0.0334	Difference in true shooting percentage over the last 20 games
away_rest_days_capped	0.0269	Number of rest days (0 - 4)
diff_roll20_3pa_rate	0.0255	Difference in 3-point attempt rate over the last 20 games
Away_b2b	0.0219	Flag for the away team having a game on the previous day
Home_b2b	0.0209	Flag for the home team having a game on the previous day

We noticed that home_indicator had no importance, which led us to the conclusion that the home interaction was learned through all the “diff” variables and that the flag wasn’t contributing anything because of that.

One important choice was keeping the tree depth very shallow (max_depth = 2). This means each tree can only learn simple splits and basic interactions between variables, rather than building very complex rules. Using shallow trees helps keep the model stable and prevents it from fitting noise in the training data.

Instead of relying on a few complicated trees, the model improves predictions gradually by combining many small, simple trees.

Predictions

For each scheduled game from March 14 to April 12, 2026, the spread prediction is generated with an XGBoost regressor using only information available before tip-off. The script builds a leak-free feature vector from each team's prior game history, including recent scoring margin, season net and defensive rating, home/road split margin, rest days, back-to-back indicators, rolling true shooting, three-point attempt rate, turnover rate, and recent head-to-head margin. Any missing feature values are replaced with the training-set medians, and the fitted XGBoost model then outputs the predicted spread, defined as expected home points minus away points.

Total Methodology

Overview and Approach

For the Total predictions, we used all regular season games in the master dataset starting from October 2021 through February 2026, giving us 6,093 games. We excluded playoff games because they tend to be played at a different pace and intensity than regular season games. All models were evaluated using 5-fold time-series cross-validation on the training set to tune parameters without look-ahead bias. The held-out test set was January 14 through February 19, 2026 (234 games), with 5,843 training games before that date. The most important rule throughout was that every feature had to be available before tip-off — we could only use information from games that had already been played.

Baseline Model

We started with a naive baseline that predicted the training set mean (227.0 points) for every game, giving a test MAE of 15.79. This was the number we needed to beat. Total had a standard deviation of 20.1 points and was nearly symmetric (skewness = 0.19).

Feature Engineering

All features were built using only past game data, always shifted by at least one game so that the current game's results were never included. Features came from two places: rolling columns already in the dataset (each team's points scored and allowed over their last 5 and 10 games, back-to-back flags, and efficiency stats), and new features we created by combining those columns.

Combined Scoring Pace

The dataset already had each team's rolling points scored and allowed over the last 5 and 10 games. We combined them into a few summary features that describe the expected scoring environment for the upcoming game:

- **avg_last5_pace and avg_last10_pace:** The sum of both teams' recent points scored and points allowed. For example, if the home team averaged 115 scored and 112 allowed over their last 5 games, and the away team averaged 110 scored and 108 allowed avg_last5_pace = 445. Higher values mean both teams have recently been in high-scoring games.
- **opp_adj_total:** (home pts_for + away pts_against + away pts_for + home pts_against) over the last 10 games. This adjusts for opponent quality rather than just looking at raw averages.
- **h2h_avg_total:** The average Total from the last 5 games between these two specific teams. Some matchups tend to be consistently high or low scoring based on how the two teams' styles match up. When no prior head-to-head history exists, we used the training mean of 227.0.
- **Back-to-back and win percentage:** Home_b2b and Away_b2b flag when a team is playing on consecutive nights. We also created both_b2b for when both teams are in that situation. For win percentage (home_winpct, away_winpct), the dataset's win/loss columns were missing for the 2021–2023 seasons, so we computed it from scratch for all seasons by tracking cumulative wins and losses from the game log before each game. It defaults to 0.5 at the start of a new season.

Per-Team Pace History

We built a per-team game log by stacking every team's appearances chronologically, both home and away. From that log we computed rolling averages and exponentially weighted moving averages (EWM) of possessions and game totals for each team, shifted by one game so the current game was never included. We then added the home and away versions together to get combined features for the upcoming matchup:

- **comb_roll20_total and comb_ewm20_total:** Each team's 20-game rolling and EWM average of game Totals, summed.
- **comb_ewm10_poss and comb_ewm20_poss:** Each team's EWM average possessions per game over the last 10 and 20 games, summed.

Feature Selection

After building candidate features, we used Lasso ($\alpha = 0.3$) to remove redundant ones. Lasso kept 10 features: comb_ewm20_total, avg_last10_pace, comb_ewm20_poss, h2h_avg_total, comb_roll20_total, comb_ewm10_total, avg_last5_pace, Home_last10_pts_against, opp_adj_total, and both_b2b. We ended up using a 13-feature set that is mostly consistent with this but keeps a few extra features.

Models Considered

We tested several model types and evaluated all of them on the same held-out test set. We used 5-fold time-series CV for hyperparameter tuning.

Ordinary Least Squares (OLS). We started with OLS to see how much of the variance in Total could be explained by a simple linear combination of our features with no regularization. We tested it on several feature sets and got test MAEs ranging from 14.69 to 14.93.

Ridge Regression. Because many of our rolling features were highly correlated with each other, we expected OLS to be somewhat unstable. Ridge regression adds a penalty on large coefficients that shrinks correlated features toward each other, making the model more stable. We tuned the penalty α using time-series CV and found $\alpha = 500$ worked best on the 13 Lasso-selected features, giving a CV MAE of 14.77 and test MAE of 14.70.

ElasticNet. We tried ElasticNet since it combines both L1 and L2 penalties. We thought this might find a slightly better feature weighting than Ridge alone. It achieved a test MAE of 14.71, which is basically identical to Ridge.

Polynomial Interactions. We wanted to test whether there were any interaction effects between features that a linear model couldn't capture. So, we generated all pairwise interaction terms from the top 8 features and combined them with Ridge regularization. This gave the lowest test MAE of 14.61, but its CV MAE of 14.89 was higher than plain Ridge, which suggests some risk of overfitting.

Random Forest. We tried Random Forest to see if a nonlinear tree-based model could pick up patterns the linear models missed. It achieved a test MAE of 14.63, but its CV MAE of 14.90 made us less confident it would generalize reliably.

Gradient Boosting. We tried gradient boosting as a more sophisticated tree method that builds trees sequentially. We ran a hyperparameter search and found the best configuration used shallow trees and a low learning rate to limit overfitting. However, the test MAE was 14.76 and CV MAE was 14.90, which were both worse than the best linear models.

MLP Neural Network. We tested a neural network to see whether a more flexible nonlinear model could find a signal that none of the other approaches captured. We used two hidden layers (64 → 32 neurons) with ReLU activations, L2 regularization, and early stopping to prevent overfitting. It was the worst performing model with a test MAE of 14.84 and CV MAE of 15.41.

Model Results

Model	Feature Set	CV MAE	Test MAE
Poly. interactions + Ridge (α=100)	Top 8 feats	14.89	14.61
Random Forest (300 trees, d=6)	13 Lasso feats	14.90	14.63
Ridge (α=500)	10 new Lasso feats	14.78	14.64
OLS log(Total)	13 Lasso feats	—	14.65
Ridge (α=500)	24-feat extended set	14.79	14.68
OLS	13 Lasso feats	14.81	14.69
Ridge (α=50)	13 Lasso feats	14.80	14.69
Ridge (α=100)	13 Lasso feats	14.79	14.70
Ridge (α=200)	13 Lasso feats	14.78	14.70
Ridge (α=500)	13 Lasso feats	14.77	14.70
ElasticNet (α=0.3, l1=0.5)	13 Lasso feats	14.79	14.71
Ridge (α=1000)	13 Lasso feats	14.77	14.71
GradBoost (n=200, lr=0.04, d=2)	13 Lasso feats	14.90	14.76
MLP (64→32, α=0.01)	13 Lasso feats	15.41	14.84
Naive (season mean)	—	—	15.79

Best Model

The polynomial interactions model had the best test MAE (14.61), but its CV MAE of 14.89 was much higher than its test MAE, which tells us it may have gotten lucky on this test window. Ridge (α = 500) on the 13 Lasso-selected features had a CV MAE of 14.77 and test MAE of 14.70. We selected Ridge as the final model because it was more consistent across folds and simpler to apply to future games.

The final model is Ridge Regression with $\alpha = 500$, trained on 13 features that are each standardized to zero mean and unit variance using training-set statistics.

Feature	Description	Std. Coef.
comb_ewm20_total	EWM avg Total, span = 20	+1.87
comb_roll20_total	Both teams' avg Total, last 20 games	+1.57
avg_last10_pace	Combined scoring pace, last 10 games	+1.10
opp_adj_total	Opponent-adjusted combined scoring, last 10	+1.10
h2h_avg_total	H2H avg Total, last 5 matchups	+0.95
comb_ewm20_poss	EWM possessions/game, span = 20	+0.91
comb_ewm10_poss	EWM possessions/game, span = 10	+0.64
Away_last5_pts_for	Away team pts scored, last 5 games	+0.58
Home_last5_pts_against	Home team pts allowed, last 5 games	+0.50
home_winpct	Home team season win %	+0.48
Home_last10_pts_against	Home team pts allowed, last 10 games	+0.44
both_b2b	Both teams on back-to-back	−0.24
Home_b2b	Home team on back-to-back	−0.06

The largest coefficients are all pace-related rolling averages, which makes sense since how much both teams have been scoring recently is the clearest signal we have for how many points the next game will produce. The back-to-back flags are the only negative coefficients, confirming that fatigued teams play slower and score less effectively.

Generating Predictions

For each of the 233 scheduled games from March 14 to April 12, 2026, we generated predictions using only information available before tip-off.

- For the rolling and EWM pace features, we used each team's full game log up to the prediction date, stacking their home and away appearances together in chronological order. Then, we computed the relevant averages over the last 10 or 20 games and summed the home and away values to get the combined feature for the matchup.
- For the dataset-based rolling features, we used each team's most recent rolling averages of points scored and allowed, computed over the same window-lengths used during training.
- For h2h_avg_total, we looked up all prior games between the two specific teams and took the mean Total from the last 5 matchups. If no head-to-head history existed, we used the training mean of 227.0.

- For home_winpct, we counted the home team's wins and losses in the current season up to but not including the prediction date and divided wins by total games played. If the team had not yet played any games that season, we defaulted to 0.5.
- For the back-to-back flags, we checked whether each team had played a game on the calendar day immediately before the prediction date.
- Once all 13 features were assembled, we standardized them using the training-set means and standard deviations. These scaling values were fixed from the training data and were not refit on the prediction games.
- We applied the fitted Ridge model to the standardized inputs to produce a predicted Total for each game.

OREB Methodology

Overview and Approach

For the OREB part of the project, the response variable was total offensive rebounds in a game, which we defined as Home OREB plus Away OREB. At first, we wanted to use the MasterDataset because it already had a lot of useful looking columns in it. After looking more carefully, though, we realized that many of those columns were based on the same game we were trying to predict. For example, some of the pace, efficiency, and rebound variables were calculated from the actual box score of that game. That would have caused leakage, because those values would not be known before tipoff. Because of that, we rebuilt the OREB modeling data from the team-level game file instead of directly using same-game summary variables from the master file.

We started with TeamStatistics.csv and kept only the seasons from 2021-22 through 2025-26. We also restricted the data to the 30 NBA teams that appeared in the prediction file so that extra rows from nonstandard teams or special games would not affect the model. After that, we kept only completed games with exactly two team rows, one for each team, and we dropped rows missing important box score columns that were needed for feature creation. Then we attached opponent values to each team row by matching the two rows from the same game. This let us create team-level rebound and opportunity variables while keeping the data at the team-game level at first.

A big goal for this section was making sure every feature was truly pregame. Since the final task was to predict future games, we only wanted to use information that would have been available before the game started. That is why the full OREB process ended up being based on lagged rolling features, matchup variables, and schedule context instead of same-game statistics.

Baseline Model

We started with a naive baseline that predicted the training-set mean TotalOREB for every game in the holdout set. This gave a holdout MAE of **4.8114**. We used this as the number to beat.
This baseline was simple, but it gave us a reference point. If the more advanced models could not clearly beat that value, then there would not be much reason to use them.

Feature Engineering

After the basic cleaning, we created the pregame features that we thought would matter most for total offensive rebounds. First, we made a missed field goals variable, because offensive rebounds usually come from missed shots. We also made a simple possessions proxy using field goal attempts, free throw attempts, turnovers, and offensive rebounds, because faster games and games with more possessions should create more rebound chances.

Next, we created offensive rebound rate, which we defined as offensive rebounds divided by offensive rebounds plus opponent defensive rebounds. We also created offensive rebound allowed rate, which measured how often a team allowed the other team to get offensive rebounds. After that, we built rest

days and back-to-back indicators by looking at each team's previous game date within the same season. Rest days were the number of off days between games, and b2b was equal to 1 when rest days were 0.

To make the variables realistic for future prediction, we built lagged 5-game rolling averages for the main team-level variables. These included reboundsOffensive, opp_reboundsOffensive, oreb_rate, oreb_allowed_rate, missed_fg, and possessions_proxy. We shifted each variable by one game first and then took a 5-game rolling mean within team and season. That way, the value for a game only used information from earlier games and never from the current one.

Once those lagged team features were created, we split the data into home team rows and away team rows and merged them back together so that each game had one row. Then we created the target variable TotalOREB, which was HomeOREB + AwayOREB. We also made combined matchup features at the game level. These included total_missed_fg_r5, total_possessions_proxy_r5, sum_oreb_rate_r5, sum_oreb_allowed_rate_r5, rest_sum, and b2b_any. We also created one interaction-style feature called cross_matchup_r5. This was calculated as the home team's recent offensive rebound rate times the away team's recent offensive rebound allowed rate, plus the away team's recent offensive rebound rate times the home team's recent offensive rebound allowed rate. We included this because total offensive rebounds in a game should depend not only on how good each team is at getting offensive boards, but also on how much the opponent usually allows.

Models Considered

For model evaluation, we used a chronological train-test split instead of a random split. We trained the models on the 2021-22 through 2024-25 seasons and tested them on the available 2025-26 games. We thought this made more sense because this is a prediction project, so the model should be trained on earlier seasons and tested on later games. We used mean absolute error as the main metric because that is also how the OREB predictions are graded in the project. We also looked at RMSE and R-squared, but MAE was the most important measure for deciding which model was best.

The first model we considered was the simple baseline. After that, we compared several different models. We fit a plain linear regression model, a ridge regression model, a lasso regression model, an elastic net model, a Poisson regression model, and a random forest model. We wanted to compare both simple linear methods and more flexible machine learning methods so we could see whether OREB behaved more like a regular regression problem or whether a more complicated model would really help. We also included Poisson regression because total offensive rebounds is a count variable, so we wanted to test whether a count-style model would be a better fit. The random forest was included to check whether a nonlinear tree model could pick up patterns that the linear models missed.

For tuning details, the ridge regression model was fit with alpha values searched over logspace(-3, 3, 50), and time-series cross-validation with 4 splits was used to choose the best alpha. The model selected alpha = 754.312006. The lasso regression model used LassoCV with alpha values searched over logspace(-3, 0, 30), 4 time-series splits, and max_iter = 20000. The elastic net model used ElasticNetCV with alpha values searched over logspace(-3, 0, 25), l1_ratio values of 0.1, 0.3, 0.5, 0.7, and 0.9, 4 time-series splits, and max_iter = 20000. The best elastic net settings were alpha = 0.237137 and l1_ratio = 0.1. The Poisson

regression model used alpha = 1.**0** and **max_iter = 1000**. The random forest model used **300 trees**, **max_depth = 8**, **min_samples_leaf = 3**, **random_state = 42**, and **n_jobs = -1**.

For all the linear-style models, we used a pipeline with median imputation for missing values and standardization before fitting the model. The median imputer was important because early-season games do not have full rolling histories yet, so some lagged features are missing at the start of a season.

Model Results

The table below shows the main holdout results for the OREB models we tested.

Model	Holdout MAE
Ridge Regression	4.5139
Elastic Net	4.5151
Linear Regression	4.5154
Lasso Regression	4.5196
Poisson Regression	4.5207
Random Forest	4.5696
Baseline Mean	4.8114

These results showed that the best OREB models were all in the regularized linear family. The nonlinear random forest model did not improve enough to justify the extra complexity, and the Poisson model also ended up slightly worse than the best linear options.

Best Model

After comparing the models on the 2025-26 holdout set, ridge regression had the lowest MAE and was selected as the final OREB model. The results were close for the best linear models, but ridge was still the winner. Based on those results, we chose ridge because it had the best prediction accuracy, but it was still easy to explain and much simpler than a larger nonlinear model.

The final model is Ridge Regression with alpha = 754.312006, trained on the OREB feature set described above. Since all model inputs were standardized before fitting, the coefficient sizes can be used to compare which variables mattered most.

Feature	Description	Coefficient
sum_oreb_rate_r5	Combined recent offensive rebound rate for both teams	+0.3357
H_oreb_rate_r5	Home team recent offensive rebound rate	+0.2616
total_possessions_proxy_r5	Combined recent possessions proxy	+0.2466
cross_matchup_r5	Home and away rebound-rate matchup interaction	+0.2210
A_oreb_rate_r5	Away team recent offensive rebound rate	+0.1889
A_oreb_r5	Away team recent offensive rebounds	+0.1667
A_possessions_proxy_r5	Away team recent possessions proxy	+0.1639
total_missed_fg_r5	Combined recent missed field goals	+0.1621
H_oreb_r5	Home team recent offensive rebounds	+0.1388

Feature	Description	Coefficient
H_possessions_proxy_r5	Home team recent possessions proxy	+0.1293

Looking at the final ridge model coefficients, the most useful variables were recent offensive rebound rate and overall rebound opportunity variables. The strongest feature was sum_oreb_rate_r5, followed by H_oreb_rate_r5, total_possessions_proxy_r5, cross_matchup_r5, A_oreb_rate_r5, A_oreb_r5, and total_missed_fg_r5. This made sense to us because total offensive rebounds in a game should mostly depend on how strong both teams have been at getting offensive boards recently, how many rebound chances the game is likely to create, and how the two teams match up against each other. The rest variables helped some, but not as much as the rebound rate and opportunity variables.

We also explored whether current-season player-level data should be added to the final OREB model. We cleaned the current 2025-26 player stats file, removed the league average row, removed traded summary rows like 2TM and 3TM, kept one latest row per player, and aggregated offensive rebounds, defensive rebounds, total rebounds, games, and minutes up to the team level. Our idea was that this could give a current rebounding strength snapshot closer to the March 14 to April 12 prediction window. However, after comparing it to the rolling team features, we decided not to include it in the final model. It added some information, but not enough to justify making the final model more complicated.

Generating Predictions

After the best model was selected, we used it to generate predictions for the future games in Predictions.csv. First, we matched the full team names in Predictions.csv to the shorter team names used in TeamStatistics.csv. Then we used the 2025-26 league schedule file to calculate exact future rest days and back-to-back indicators for all prediction games between March 14 and April 12. For each team, we found the most recent scheduled game before the prediction date and used that to compute rest_days and b2b.

After that, we created the future game feature rows by taking each team's latest available rolling OREB features from the cleaned historical team feature table. We merged the latest home team features and latest away team features into each prediction row, then rebuilt the combined features such as total_missed_fg_r5, total_possessions_proxy_r5, sum_oreb_rate_r5, sum_oreb_allowed_rate_r5, cross_matchup_r5, rest_sum, and b2b_any. This gave each future game the same feature structure that the ridge model had used during training.

Finally, we passed those future game feature rows into the fitted ridge regression model and generated a predicted TotalOREB value for each game in Predictions.csv. We rounded the predicted OREB values to the nearest tenth and wrote them back into the OREB column of the prediction file. In this way, the final predictions came from the same model and same feature definitions that were used in the historical testing stage. Overall, we think the final ridge regression model was the best choice for OREB because it clearly beat the baseline, slightly outperformed the other candidate models, used only realistic pregame information, and was simple enough to explain clearly.